DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Physical Embodiment in VR: Interchangeable Web-Based Modules using Ubi-Interact

Leonard Goldstein

DEPARTMENT OF INFORMATICS

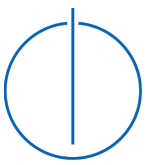
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Physical Embodiment in VR: Interchangeable Web-Based Modules using Ubi-Interact

Physikalische Verkörperung in VR: Austauschbare Webbasierte Module mithilfe von Ubi-Interact

Author:	Leonard Goldstein
Supervisor:	Prof. Gudrun Johanna Klinker, Ph.D.
Advisor:	Sandro Weber, M.Sc.
Submission Date:	June 15, 2022



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, June 15, 2022 Leonard Goldstein

Acknowledgments

I would like to thank my advisor Sandro Weber for his tireless and insightful weekly advice, as well as for providing access to hardware.

Furthermore, I would like to express my gratitude towards my supervisor Prof. Gudrun Klinker for the opportunity to write my Bachelor's thesis in such an exciting area and for her optimistic encouragement at my kick-off presentation.

Last but not least, I am grateful for my family's and friends' sympathetic ears and patience in times of stress and uncertainty. Especially, I thank Filip Skubacz for his occasional reminders to remain diligent.

Abstract

Physical embodiment in VR enables users to easily interact with a virtual world while realistically adhering to the physical framework of that world. Due to the countless amount of different hardware and software in the VR area, this process can be most efficiently realized using a modular, network-based approach. An existing implementation that uses the framework Ubi-Interact in Unity with C# already shows that it is possible to implement such software. A new implementation uses Ubi-Interact in the web browser with Babylon.js and WebXR. This implementation uses the same structure as the Unity implementation and is, to some extent, compatible. Modules of both implementations can be replaced with their counterparts of the other implementation, demonstrating that Ubi-Interact is a suitable framework for physical embodiment. However, the web platform still has few suitable libraries, such as in the areas of Inverse Kinematics, which leads to incomplete and incorrect data. Some insights are found regarding precision and performance using an additional Ubi-Interact-based module for evaluation. In the outlook, further improvements and questions surrounding the implementation and the topic of physical embodiment are proposed.

Contents

1.Introduction

Many kinds of devices and technical approaches related to Virtual Reality (VR) and Augmented Reality (AR) exist. This variety is a challenge for developers who want to create applications compatible with a vast range of hardware. It also makes it hard to scientifically explore issues from this space because implementations cannot be easily compared or combined. To solve these and other issues, S. Weber et al. developed Ubi-Interact [1]. On a high level, it is a framework for coordinating the communication of application modules in real-time over a network.

One fascinating research topic in VR is physical embodiment. In many VR applications, the user wants to interact with the virtual world and possibly even other users in an immersive way. For that, users need a virtual counterpart – an avatar closely resembling the user's posture in real-time [2].

In the last two decades, web browsers have evolved from displaying textual content to being hosts of many kinds of interactive applications. With the availability of WebGL and WebXR, modern browsers have become an immersive platform of interest for VR/AR developers [3]. Nevertheless, browsers run interpreted code that must work reliably on a vast range of devices, so its performance is worse than that of native applications.

This thesis will examine these three aspects by examining physical embodiment in VR in web-based environments using Ubi-Interact. It will analyze the similarities and differences between an own web-based implementation and an already existing implementation written in C#/Unity [4].

2.Previous Research

2.1. Definitions

2.1.1. Virtual Embodiment

Embodiment can be understood both as a practice and as a sense.

As a practice, the term *embodiment* is broad, as it is generally used to indicate that something has certain qualities [5]. In the context of VR, embodiment may refer to *virtual embodiment*. *Virtual embodiment* is often narrowed down, describing only the process of virtual replication of a physical person's body [6]. The term *digital twins* is used when including virtual replication of non-animate objects[7].

Embodiment can also represent a sense. According to Kilteni et al., experiencing a *sense of embodiment* towards a body depends on three aspects: whether the user feels that he is inside the body, feels to be an agent of the body, and whether he perceives it as his body [8]. Each aspect can be evaluated separately, for instance, using specific questions in questionnaires. A *full sense of embodiment* can only be experienced whenever all of the said aspects are felt at full intensity. Similar concepts to sense of embodiment exist that highlight other aspects and are more dependent on the virtual environment rather than just the body itself. These include the *sense of presence* or the *Being-There* as well as the more general term *immersion* [9, p. 392].

2.1.2. Embodiment and Re-embodiment

The meaning of the term *embodiment* in contrast to *re-embodiment* was discussed by Besmer: For full *embodiment*, the user needs to receive feedback beyond visual representation, such as tactile feedback [10]. S. Weber et al. follow this distinction in research that this thesis builds upon [2]. While this work does not explicitly deal with such additional feedback, extending the implemented software components, for example, with tactile feedback, would be possible. In the context of modular

solutions such as the one presented here, the exact implementation of the interface that interacts with the user is dependent on the specific use case. It could very well provide additional feedback to the user. From a network communication perspective, the data sent from a user to a physical avatar and the data processing in between would not differ. Given the modular nature of the process used in this thesis, instead of making assumptions about how the components introduced here are (not) going to be used, solely the term *embodiment* is used, even though the demonstration implementation only uses visual feedback.

It is important to note that other definitions of *re-embodiment* in the context of human-computer interaction define it as the movement of social presence between multiple agents. An example of this context is home assistants, which are embodied in different devices [11]. This thesis does not follow this definition because the concept of embodiment in that work does not refer to *virtual embodiment* as it is defined here – it also includes virtual entities that are embodied physically.

2.1.3. Physical Embodiment

The virtual avatar's pose does not only have to resemble the user's pose closely, but it also has to behave physically correct in the virtual environment to support *immersion* in *virtual embodiment*. Such realistic behavior can be achieved by applying rules of physics to an avatar, such as gravity, by using a physics engine. *Virtual embodiment* processes that use such techniques could therefore be called *physical embodiment*. "Physical" in this definition does not oppose the concept of virtuality because it does not stand for the actual existence of the avatar but because the virtual avatar behaves like a physical person in its respective environment. However, when leaving the field of *virtual embodiment* and including robotics, *physical embodiment* can be achieved with robots that resemble a user's pose in the context of real physics. The process would be similar but is not in the scope of this work.

2.2. Process of Physical Embodiment

A physical embodiment implementation can be split into tasks from different disciplines:

Due to the lack of complete input data to represent a whole avatar, Inverse Kinematics (IK) can be used to determine a full pose from so-called end effectors. A myriad of IK algorithms exist, with varying computational effort and in various fields, such as robotics, animation and games [12].

Virtual avatars and objects have to react to interactions similar to real ones to give the user the option to interact with his surroundings immersively. To obtain this, real-time virtual physics can be used as described in the previous section. In this area, much research has been conducted, for example, in the context of simulation for robotics [13] and character animation [14].

Unifying all of these aspects, Weber et al use a 3-stage process for physical embodiment [1]. The following stages are executed repeatedly and in parallel:

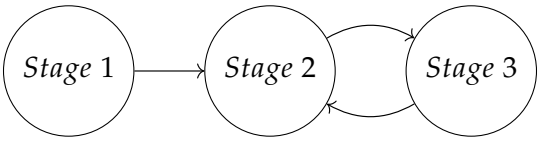


Figure 2.1.: Communication between physical embodiment stages

- **Stage 1**: Partial pose data from a user is obtained, for example, from hand tracking devices. The data is sent to stage 2.
- **Stage 2**: Using data from stage 1 and IK, a full pose is calculated to resemble the actual user's pose. The linear and angular velocities required to adjust a physical avatar are calculated using additional data from stage 3. These forces are being sent back to stage 3.
- **Stage 3**: The forces resulting from stage 2 are applied to a physical avatar. The new pose is sent back to stage 2.

2.3. Networking and Ubi-Interact

The stages above are relatively independent and could even be distributed across multiple hardware devices. Distribution can potentially be helpful to support a wider variety of devices, send and process data efficiently in multi-user scenarios, or make calculations on different hardware depending on dynamic performance and delay requirements. Especially in the context of mixed reality, given that there are a lot of different standards and devices, a framework that manages the real-time network connections is required.

Ubi-Interact is such a framework. *Ubi-Interact* consists of a master node and client libraries, which applications can use to communicate via the master node.

It abstracts endpoints using so-called *devices*. *Devices* can but do not have to represent real devices. A *device* groups multiple *components*. Components define whether they publish or receive data and a communication *topic*. *Components* may also define *tags* so that they can be found by other *Ubi-Interact* applications that may not know the exact topic name.

Another similar concept in Ubi-Interact is *processing modules*. *Processing modules* also in- and output data but are defined more abstractly so they can be re-used. Similarly to *devices* and *components*, assigning *tags* to *processing modules* is possible for better discoverability.

When developing network applications that should handle large amount of data, choosing an efficient data format for transmission is crucial for performance. Ubi-Interact uses so-called Protocol Buffers (protobuf). Protobufs minimize data objects by using integer indices instead of the original object's keys. When both peers know the protocol buffer definition, they both can translate inbetween indices and keys and thus communicate [1].

2.4. Physical Embodiment Implementation in C# / Unity

S. Weber implemented a working implementation of the physical embodiment process using the process described above with Ubi-Interact and C# / Unity. While

still being in development due to the ongoing research and changes in Ubi-Interact, it serves as a functioning demonstration of the feasibility of the process.

The Unity implementation runs all three stages on one physical device. By selectively removing modules from the source code, it can be shown that the content can be split across multiple application instances in a network.

Inverse Kinematics were implemented using the built-in Inverse Kinematics Animation controller. The documentation does not disclose which Inverse Kinematics algorithm is used [15].

Similarly, the built-in 3D physics engine was used, which is an integration of the Nvidia PhysX engine [16].

The Unity implementation has VR support in theory for demonstration purposes but works mainly by emulating inputs by reading from animations rather than using real controllers. Using animations simplifies debugging because of the lack of necessity of using hardware and broad movements during development.

2.4.1. Resulting Protocol

Formal documentation of the data sent and received in the Unity implementation is not available yet. From the implementation, the following standards could be deduced that can be used in other implementations to guarantee compatibility:

Stages 1 and 3 are represented using a device. Each incoming and outgoing connection from these stages represents a component. Stage 2 is represented using a processing module. Each connection between the stages is communicated using a distinct topic. Because all transmitted data is either positional, rotational, or both, the defined protbuf format `ubii.dataStructure.Object3DList` as defined in ubi-msg-formats [17] can be used for all topics.

The following IDs may be expected in objects on `/avatar/ik_targets`: `HEAD`, `VIEWING_DIRECTION`, `HIP`, `HAND_LEFT`, `HAND_RIGHT`, `FOOT_LEFT`, `FOOT_RIGHT`.

The other topics allow 52 IDs, each referencing one bone. The list is included as an Appendix. Many of the permitted values are effectively not in use because the data

from `/avatar/ik_targets` is insufficient to calculate exact poses for some bones, such as fingers.

Table 2.1.: The following topics names and tags should be used.

Topic suffix, prepended with device id	Involved stages	Publisher Tags	Subscriber Tags
`/avatar/ik_targets`	1 to 2	avatar, user tracking, ik, targets, ik targets, inverse kinematics	*n/a*
`/avatar/target_velocities`	3 from 2	*n/a*	avatar, bones, control, velocity, linear, angular
`/avatar/current_pose/list`	3 to 2	avatar, bones, pose, position, orientation, quaternion	*n/a*

2.4.2. Open Questions from Unity

The Unity implementation proves that physical embodiment in VR over a network is possible using the three previously discussed stages. However, it alone leaves unanswered questions.

For example, the concept of modularity should be explored further in this context. As mentioned before, the project in its current state is meant to run on one device, in one application. How well does running different stages on different physical devices over a network work, to what extent is that feasible, and does it compromise

the user experience?

Another aspect in the context of modularity is whether the modules are replaceable. Given a different implementation that uses the same protocol, can stages from such an implementation be used as a drop-in replacement of stages in the unity implementation? Furthermore, in reverse, can stages of the unity implementation be used to replace stages from a different implementation? If this is not possible due to the necessity of tweaking values for every configuration or due to unfixable incompatibilities, one would have to conclude that the modular concept of Ubi-Interact failed on the topic of physical embodiment in its current form.

Also, as of now, there is no data on the accuracy and performance of the implementation. Such data needs to be measured as a basis for further comparisons.

The effect of mainly using animations for testing needs to be discussed. Actual pose data might contain more noise than the one obtained from animations or might differ in other, unforeseen ways.

Apart from the technical aspects above, scenarios in which physical embodiment over a network is required must be discussed to put the available information into context.

3.Related Work

This thesis and its underlying work are not the first work that combines the fields of VR, networks, and embodiment. Existing research includes many application areas, two of which will be highlighted in the following sections.

3.1. Robotics

Many single-user appliances of embodiment and networks can be found in robotics, where humanoid and non-humanoid robots can re-enact users' poses. For example, VR controllers could be used in manufacturing to control robot arms.

Brizzi et al. examine the effects of Augmented Reality for teleoperated industrial assemblies: Accuracy and the sense of embodiment differ depending on the kind of feedback that the user receives. Especially influential for increasing the sense of embodiment was displaying a virtual model of the robot that the user embodied. Displaying much information increased the accuracy of task execution, but the users felt a weaker sense of embodiment. In general, it was concluded that AR could help simplify learning the teleoperation of a robot [18].

Not only in manufacturing but also in surgery, teleoperation can be used. Laaki et al. prototyped a VR remote surgery solution that can be used even when only 4G mobile internet is available. Because of low bandwidth, instead of showing the user an exact patient representation, a digital twin was used [19]. In this case, both surgeon and patient are embodied – the surgeon in the surgery robot and the patient virtually.

3.2. Multi-User Scenarios

The idea of using VR for more immersive communication has been discussed at least since the 1990s, albeit prototypical implementations were not particularly immersive [20]. Many problems that implementations of conferencing solutions

had could already be solved due to better hardware; for example, [21] shows that users felt present during usage of a conferencing solution with VR a headset.

With over 100,000 reviews on the video game platform Steam, a popular VR platform is VRChat [22]. It enables users to embody an avatar of their choice in virtual Reality and interact with other users' avatars. The effects of embodiment in VRChat have been studied from a psychological and social standpoint: Research by Rzeszewski et al. indicates that users felt emotionally attached to virtual places and that during lockdowns in the COVID-19 pandemic. For its users, VRChat served as a substitute for non-digital social interaction [23]. There are different interpretations on whether the free choice of an avatar in such a scenario support dissolving societal norms, primarily cultural and gender norms [24, p. 33], or whether the opposite is the case and users tend to embody highly stereotypical avatars [25].

4.Goal of this thesis

4.1. Problem Statement

The work in this thesis consisted of two parts: implementing an alternative version of the physical embodiment process (the "Web implementation") and evaluating aspects of both implementations. It examines to what extent physical embodiment is possible in the browser and whether Ubi-Interact can be used to reach cross-implementation compatibility.

4.1.1. Implementation Requirements

The goal was to create a version that runs in a web browser. The selected framework was Babylon.js for its WebXR support. A basic setup using Babylon.js was created before this thesis, consisting of an animated avatar.

The web implementation should use the same communication standards as the Unity implementation for compatibility. Rather than being a direct port, considerations about chosen algorithms should be done according to the platform, which, in some aspects, might have a more abstracted and thus potentially more limited and slower access to system resources.

Further modularization is required to test efficiently and reuse code in further research or non-scientific contexts. The goal was to make it possible to run every stage independently by publishing them as entirely independent modules.

4.1.2. Evaluation Requirements

Any of the eight combinations of implementations of each stage should be tested to obtain information on compatibility, time performance, and positional accuracy. The results should be evaluated against each other and generally established rules for real-time interaction.

Because some results would not be precisely measurable by the eye of sight, the evaluation process contains implementing an additional module that collects and saves data both from the input pose and the resulting physical avatar's pose.

4.2. Structure

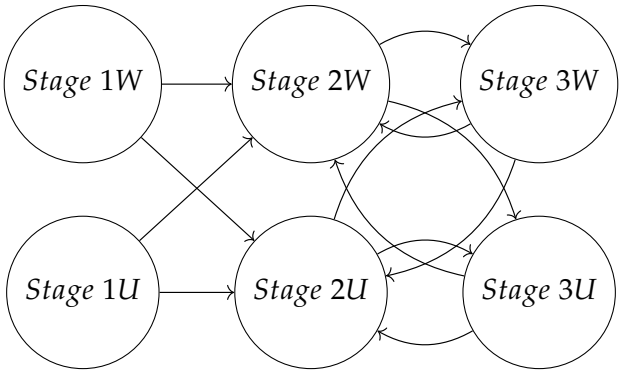


Figure 4.1.: Possible communication configurations between physical embodiment stages across the implementations Web (W) and Unity (U). There are $2^3 = 8$ possible configurations.

4.2.1. Stage Modules

As mentioned before, the three stages shall be implemented as modules. Those can be published separately in the npm registry. They further abstract the communication with Ubi-Interact for this specific use case – a developer would not have to have a deeper understanding of Ubi-Interact concepts: They could initialize each step with one line of code. These modules should work as standalone applications with graphical configuration interfaces for demonstration purposes and as an API.

- **Stage 1** publishes a simple movement by default. When used as API, custom pose data, such as input from tracking devices, can be passed in.
- **Stage 2** uses a general-purpose IK library from another module and calculates

the pose for a human skeleton.

- **Stage 3** publishes a standard pose by default and does not consider the velocities. A physics engine can be used by configuring the API to callback a velocity receiving function and publish a different calculated pose.

4.2.2. Main Application

Additionally, a main application, also referred to as Babylon.js implementation or demonstration application, should be implemented. The main application should be based on the 3D engine Babylon.js. It connects stage 1 to the debugging animation or actual input devices, displays the pose of all three stages, and applies physics using a physics engine for stage 3.

4.2.3. Evaluation Module

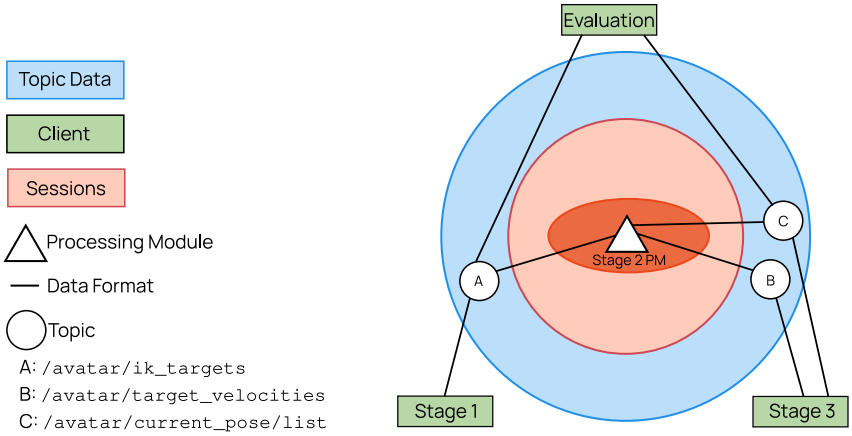


Figure 4.2.: Integration of an evaluation module in the physical embodiment process. To fully compare input and output poses, stage 1 needs to be modified to publish the full pose, not just IK targets. Without modification, only target positions can be compared. Based on [1].

The evaluation module should work similarly to the stage modules, but the intended way to use it is the graphical interface. Two topics can be given to the evaluation module, which will then record everything published.

5.Implementation Process

5.1. General Process

The plan for implementation was to start with the basic communication structure in the stage modules. This process was chosen to start broadly, which would help to identify problems early. To make the modules communicate correctly, mock data was used. After those modules were written, the next step was connecting them with the Babylon.js project. Given that the structure was set up, the mock data could be replaced at this point. The last step was to implement the evaluation module and to modifying parameters according to testing results.

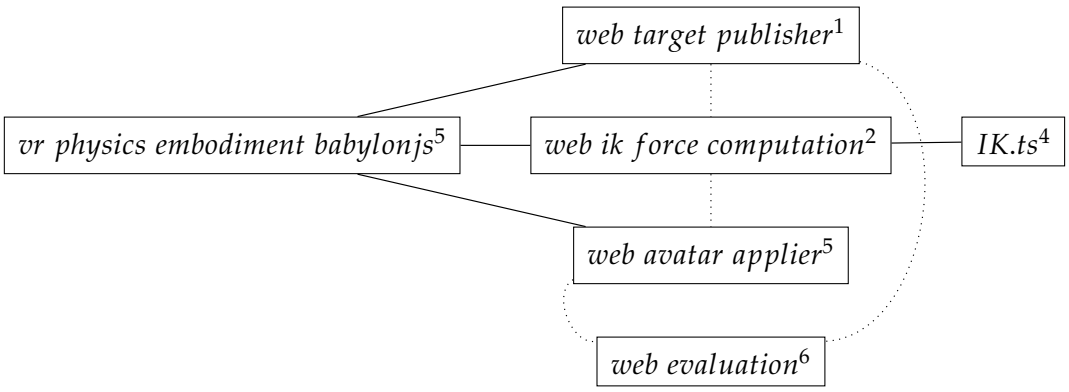


Figure 5.1.: Overview over implemented projects. Dotted lines represent communication via Ubi-Interact.

5.2. Usage of TypeScript

TypeScript was chosen as the programming language for the projects. TypeScript is a superset of JavaScript which is compiled to JavaScript. It checks type safety at compile time. Given the way Ubi-Interact's library ubii-msg-format, which uses protobuf, works, publishing data in the wrong format may not lead to error messages, but to empty published objects [17]. Having type evaluation at compile time helped speeding up the process of development.

For using TypeScript efficiently with this existing library ubii-msg-formats, type definitions had to be generated, which were added in the respective repository. Future development in TypeScript will be easier due to the included definition files, and even JavaScript developers can profit from this, because modern code editors like Visual Studio Code show better suggestions in JavaScript projects using TypeScript definitions [26].

5.3. Usage of Ubi Interact

5.3.1. Browser Limitations Regarding Graceful Exits

During the process of closing a native application, it is common that applications perform actions such as closing open connections. For some of the most common ways of leaving a website, like closing the tab, the possibilities perform such actions are severely limited. More specifically, a web application cannot send data to a server signaling that it was closed. In the context of Ubi-Interact that means

[1] Represents stage 1 of the PE process. Source code and online instance available at `https://github.com/goldst/ubii-web-target-publisher`

[2] Represents stage 2 of the PE process. Source code and online instance available at `https://github.com/goldst/ubii-web-ik-force-computation`

[3] Represents stage 3 of the PE process. Source code and online instance available at `https://github.com/goldst/ubii-web-avatar-applier`

[4] Inverse Kinematics library. Source code available at `https://github.com/goldst/IK.ts`

[5] Main project, in which all components are used. Source code available at `https://github.com/SandroWeber/ubii-vr-physics-embodiment-babylonjs/tree/feature/leonard`

[6] Evaluation module. Source code and online instance available at `https://github.com/goldst/ubii-web-evaluation`

that, even though methods for disconnecting exist, it would be common to exit without disconnecting. After a short period of time, The Ubi-Interact master node notices that no response from the client is given and performs necessary cleanup by itself. Nevertheless, there are scenarios in which this is not sufficient and has to be accounted for from the client side: The development server of the main application reloads the page when the code is changed. After such a reload, in case that the old components would not yet be de-registered and the new ones would not be finished registering, modules would connect to old topics that are not in use anymore. With some additional code, such problems were be avoided.

5.3.2. Devices versus Processing Modules

Devices and processing modules are similar concepts: Both can receive and publish data. However, the areas in which they are used usually differs. Processing modules are defined in a more abstract manner and thus can be mapped to inputs and outputs after establishing devices.

In the Unity implementation, stage 2 is implemented using a processing module. The Ubi Interact C# libraries allow running processing modules in a dedicated node – which is implemented as part of the Unity application.

In JavaScript, there were two options:

1. A client node can send a stringified JavaScript function to a master node. The master node can then run the code. This is the way that an example contained in the web browser library processes data.
2. A separate node can run dedicated processing modules, similar to the C# version.

The Ubi Interact library for web browsers only implements option 1. Option 2 is implemented in the ubi interact library for Node.js servers and is not usable in web browsers due to storage functionality that requires direct access to the file system.

One of the main objectives of the JavaScript implementation is to examine the ability of browser environments to execute the whole process of physical embodiment. None of the available options would execute stage 2 code in the browser itself. Thus,

originally, stage 2 was implemented using a device in the JavaScript implementation. This implementation worked, but using a device instead of a processing module had several disadvantages. This stage 2 implementation has to listen to available topics first and can only then create the device. Also, it is potentially less performant, because the code would run on the main thread.

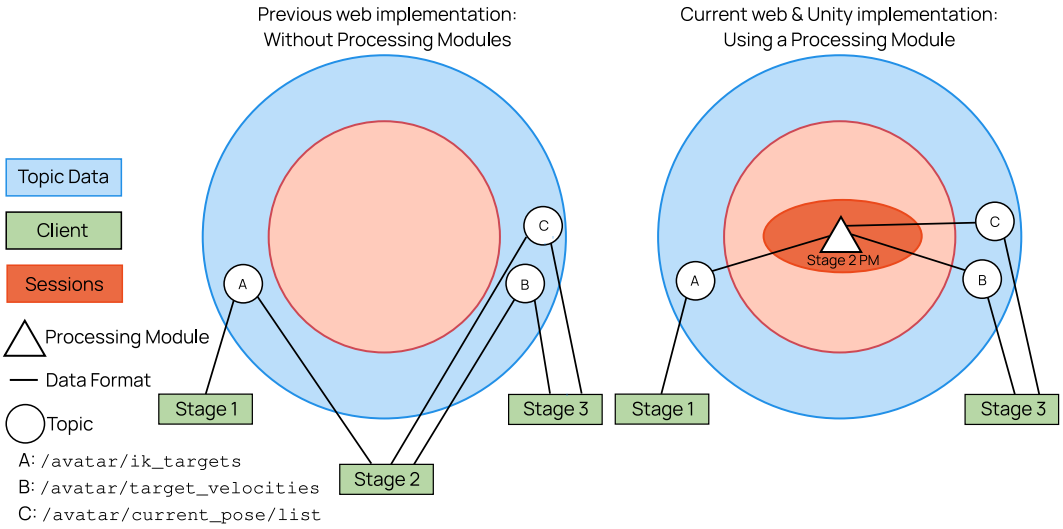


Figure 5.2.: Comparison of the original implementation with the implementation using a processing module. The in- and outputs of each stage remain unchanged, making all versions compatible with any other implementation. Based on [1]

Those issues could possibly be resolved by further adjusting the code. Instead of that, the decision was made to adapt the option 2 code to the web browser library. For that, some functionality had to be removed, such as permanent storage. This solution uses the workerpool library. The workerpool library allows running functions in separate threads – in this case, this would be used for the processing function.

Limitations of the TypeScript compiler and the used bundler webpack make it impossible to import all required dependencies such as the inverse kinematics library in a web worker directly [27], thus the current version still runs on the main thread, which might negatively affect the performance of the stage 2 execution. By configuring an additional build target in a potential future version, it would

be possible to include the dependencies in the worker function and thus run the worker more efficiently. However, this would potentially complicate displaying the stage 2 avatar.

In the future, depending on the further direction of the Ubi Interact project, it should be considered to merge the browser compatibility of the adapted version with the original Node.js implementation, so that the Node.js library can be used both for web client nodes and separate server-side nodes. For permanent storage in browser environments, the File System Access API [28] might become an option, which is only partially available for about 40% of internet users at the time of writing [29] and will likely not be fully adapted by all major browsers in its current state, as Mozilla considers the feature harmful [30].

An easier solution could be to add the modified parts to the browser library only. This would still require some refactoring and design decisions, because the currently used version in the stage 2 module is partially converted to TypeScript.

5.4. Implementation of Inverse Kinematics

For stage 2, a Inverse Kinematics library had to be chosen. The following requirements were necessary:

- The library has to be performant enough to run in real time on low-end devices. More than half of the worldwide web traffic is generated through mobile devices [31], so it is a reasonable consideration not to exclude them on a library level already.
- The library should be documented, actively maintained and easy to install.
- The library should not have dependencies to 3D engines such as THREE.js (or Babylon.js). Module two is specifically designed to work with every engine and potentially having to load two large libraries due to using another one would cause unnecessary overhead – especially in scenarios where they are downloaded from the internet on each reload, which would be the case in many usage scenarios.

- The library should allow long enough chains and bone constraints for human bodies.

No library met all requirements. Some observed options were:

- **The built-in IK functionality in Babylon.js** [32] doesn't support longer chains than chains consisting of 2 bones, constraints or operating separately from Babylon.js.
- **BussIK-js** [33] uses the Selectively Damped Least Squares algorithm, which supports multiple end effectors and converges fast [34]. However, this implementation takes up to multiple seconds to converge in Firefox, while the performance in Chrome is better. This library does not support joint constraints.
- **closed-chain-ik-js** [35] is well-documented and supports joint constraints based on limiting degrees of freedom. However, it lists THREE.js as a peer dependency. When testing the online demo, this library appeared to take longer to compute a stable state than other libraries and tended to diverge or even lock in wrong states that it cannot escape from afterwards, which was the reason that this library was not chosen.
- **Fullik** [36] has no documentation apart from uncommented examples and was not actively maintained at the time of the start of this thesis, but is a port of the well-documented Java IK library Callico, using the FABRIK algorithm [37, 38]. It does require THREE.js as a dependency. It is not uploaded as a node module to the npmjs package registry and did not run without modifications in the Babylon project. Nevertheless, Fullik seemed to be the best contender due to the lack of other documented libraries with all features and a suitable performance..

To use the code from Fullik, a fork named IK.ts was created: IK.ts is ported to TypeScript. It is using the class notation instead of the object notation that Fullik made use of[1]. It also doesn't require THREE.js. THREE.js was previously used only in a few points in the library – however, removing it makes the demos dysfunctional

[1] After the fork and modifications were already mostly implemented, Fullik introduced similar modernization changes in May and June 2022.

and connecting it to a THREE.js project more complex. A potential improvement of IK.ts would be adding adapter classes for popular 3D and 2D libraries, but this is not in scope for this thesis. Furthermore, some documentation was added.

Other than that, the code was not modified, meaning that the available constraints and limitations are the same as in Fullik.

5.4.1. Excursion: The FABRIK Algorithm

A kinematics algorithm calculates an end point of a chain, given a start point, lengths of chained segments and angles between the segments or segment positions.

Inverse kinematics algorithms do the opposite: Given a number of chained segments n, segment lengths $l_1 \dots l_n$, a start position S and a target position T also known as *end effector*, an inverse kinematics algorithm calculates a possible combination of segment end points $P_1'' \dots P_n''$ so that the chain reaches the end effector as close as possible. Some inverse kinematics algorithms also support start positions $P_1 \dots P_n$ as an input to support smooth movements.

Given differing requirements in fields such as robotics, game development and simulation a myriad of inverse kinematics algorithms exist – some of which are geared towards precision, others leaning towards fast execution time. A popular algorithm is FABRIK by Aristidou et al. [38]. FABRIK is a comparably simple and efficient heuristic algorithm. An advantage of FABRIK over other algorithms is that the needed precision and time can be adjusted using a set margin of error. In a case where the hardware specifications might differ from case to case, such as in the scenario of this thesis, such a margin could be re-adjusted during runtime.

FABRIK iterates through the list of segments repeatedly, adjusting their direction by rotating them towards the next segment. By repeatingly adjusting the rotations in both backwards and forwards direction, the last segment will move towards the end effector. As soon as the last end point is close enough to the end effector, the algorithm terminates.

The algorithm can be extended to support not only linear chains, but also tree structures.

Besides these features, FABRIK can be extended by adding constraints to the segment's rotations. Available options differ depending on whether a 2- or more-dimensional calculation is being performed [38].

Simplified description of the FABRIK algorithm

1. If goal is as far or further away than sum of lengths ($\sum l$), fully elongate the segments on a straight line to goal. End.
2. Set $P_0 := S$
3. Set $P'_n := T$
4. **Backwards algorithm**:
   a) From last calculated P'_i, calculate P'_{i-1} by going to the direction of P_{i-1}, but only as far as the length l_i
   b) Repeat step 4 until P'_0 is found

   0
5. Set $P''_0 := S$
6. **Forward algorithm**:
   a) From first calculated P''_i, calculate P''_{i+1} by going to the direction of P'_{i+1}, but only as far as the length l_{i+1}
   b) Repeat step 6 until P''_n is found
7. repeat steps 3 - 6 until P''_n is close enough to T (margin of error)

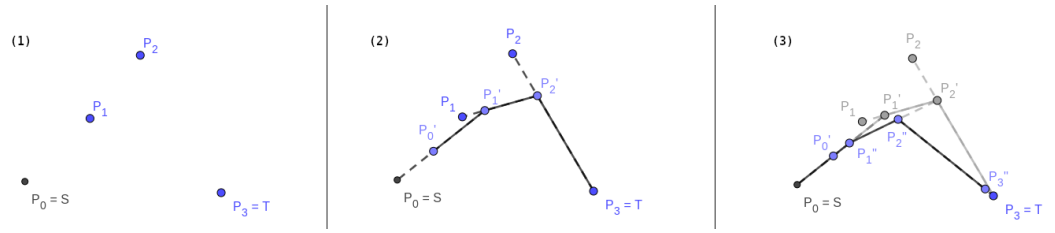


Figure 5.3.: Visualisation of a first iteration of the FABRIK algorithm: (1) Input data. (2) Application of the backward algorithm. (3) Application of the forward algorithm.

5.4.2. Problem: Insufficient Joint Constraints

In 3D mode, Caliko and therefore Fullik and IK.ts support ball (`BALL`) and hinge (`GLOBAL_HINGE`, `LOCAL_HINGE`) joint constraints[2].

Ball joints allow a bone to tilt and rotate freely at its start point, as long as they do not tilt further in any direction than to a predefined angle in relation to the previous bone.

Hinge joints allow a bone to only tilt in one direction up to a specific angle, which makes it a potential solution for bones such as lower legs, which mostly only tilt for- and backwards at the knee.

When using global hinges, the direction is set in relation to world space. For a virtual avatar that can in theory move in any direction, fixing rotations to a world space direction is not feasible.

When using local hinges, the direction is set in relation to the previous bone. If any previous bone is not constrained using a hinge, that previous bone can have any rotation, so that the hinge behaves, from a world perspective, like a ball joint – it can tilt to any direction. This restricts the usefulness of hinge joints for human bodies.

Therefore, the given joint types are not versatile enough to ensure realistic poses for human skeletons.

5.4.3. Problem: Missing Bone Rotation Output

In the previous section, it was described that in most cases, any bone can rotate freely around its own axis. For this reason, Caliko and the derived implementations do not calculate an actual rotation in these cases. Even when local hinges are being used, the rotations that are calculated in the process of solving are not preserved for the library user, who only has access to the start and end points. In many cases, those rotational values are not required, such as when modeling a spider

[2] Basebones have slightly different available constraints, these are not discussed in this problem description for the sake of simplicity.

or a shower hose, but especially in cases like human head and chest rotation, just picking one possible direction is not enough.

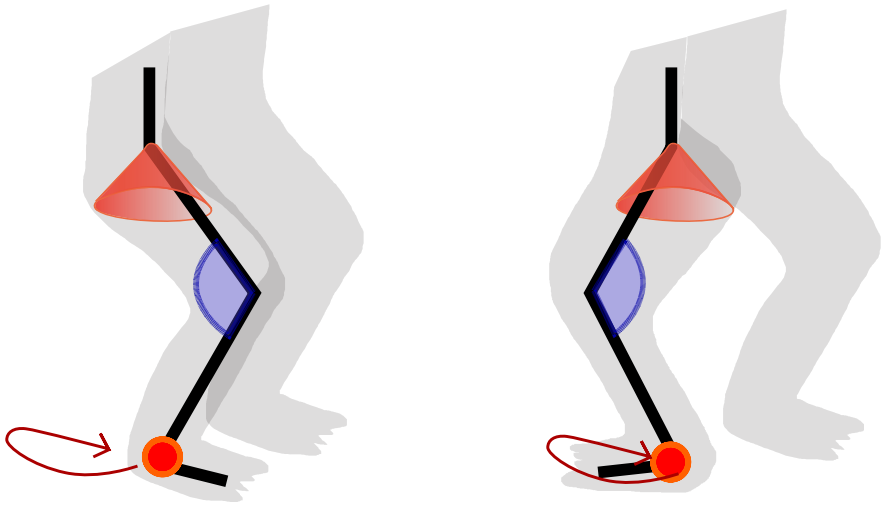


Figure 5.4.: Example of unrealistic poses as a result of insufficient joint constraints. The upper leg uses a ball joint (orange cone), so that it can move in any direction, not just forward and backward. The lower leg uses a hinge joint (blue) to resemble the way a knee works. Because the upper leg can also rotate, the hinge ends up being on the wrong side after some movement of the target (red ball).

Nevertheless, it partially had to be done to achieve results. The following formula was used to determine the quarternion rotation Q given the start and end point of a bone P_1 and P_2 and a normalized reference vector $\hat{v}$:

$$Q = (1 + \hat{u} \cdot \hat{v},\ \hat{u} \times \hat{v})$$

where $\hat{u}$ is the unit vector with the direction from P_1 to P_2 [39].

Significantly improving the IK library in these aspects was not in the scope of this thesis – refactoring it to not use THREE.js was already more work than what was planed originally. For efficiently conducting further research in the areas of physical embodiment in browsers, it would be a first important step to improve

this aspect.

Another option would be to reimplement stage 2 using a different library like closed-chain-ik-js [35], which was considered the second-best option upon selection. It would be necessary to separate the module from the THREE.js peer dependency. The available constraints seem to be more suitable for human IK, but it would have to be tested whether the performance requirements would be met.

5.5. Physics Engine

Because physics in many use cases are tightly coupled with the visual representation and the environment in which the user plans to use the physical embodiment, the decision was made not to implement physics as a part of the stage 3 module, but to implement it in the main project. Similarly to IK, Babylon offers built-in methods for physics, although built-in physics support is much more mature than built-in IK support in Babylon. Babylon offers support for the physics engines Cannon.js, Oimo.js and Ammo.js. Because stage 3 only requires basic features, all three libraries can be used interchangeably. Ammo.js is included in the implementation, Cannon.js and Oimo.js are downloaded as well when connected to the internet. The debugging GUI allows switching the engine during runtime (See subsection 5.7.3).

5.6. WebXR Support

To show the theoretical possibility of using VR inputs and outputs, WebXR support was added using the Babylon.js WebXR feature. The optional use of a WebXR compatible headset and two controllers as hand targets can be enabled using a button. This feature was tested using Mozilla's WebXR emulator [40]. While this proved that the feature is functional, to use it in real scenarios, values might have to be adjusted and an option for calibration would be necessary to make the controls more realistic.

5.7. User Interfaces

5.7.1. Stage Module GUI

To demonstrate that every stage can be used on its own, without the main project, each stage contains a user interface that can be used to configure and start the stage independently. The configuration options in the user interfaces are more limited than when using programmatic configuration. For example, it is not possible to pass callbacks into the module. A list of available configuration options can be retrieved from the screenshots in the appendix.

5.7.2. Evaluation Module GUI

Similar to the stage module GUIs, an user interface for the evaluation module was implemented. It does not serve as a demonstration interface like the stage module GUIs, but is the intended way to use the module. It supports export of the collected data in JSON format.

5.7.3. Main Project Debugging GUI

Running the project in differing configurations, restarting certain application parts and adjusting parameters was required during development and evaluation. To make this process easier, a user interface was added that allows to control these aspects of the main application.

Table 5.1.: The following options are present in the debugging GUI.

Option	***Default value*, Description**
`skipUbii`	*Disabled*. Enabling will skip the use of Ubi-Interact and will route all data internally.
`runStage1`	*Enabled*. Allows to disable the respective stage 1 when it is supposed to run in a different implementation.
`runStage2`	
`runStage3`	
`useXR`	*Disabled*. Stops animation and displays button to enter the XR mode. XR mode works with two controllers, tested using the WebXR API Emulator [40]
`selectedAnimation`	*Walk*. Allows the user to chose between a walk, idle or run animation, and manual mode.
`animationSpeed`	*1*. Slows down or speeds up the animation.
`engine`	*Ammo*. Choice of physics engine. Ammo is included in the installed packages, the other engines are fetched from the internet during runtime if a internet connection is available.
`masses`	*0.01*. The weight of each individual bone.
`Gravity`	*(0, -9.81, 0)*. Simulated gravity for the weighted bones. The default value is equivalent to the gravity on earth.
`constraintDistances`	*Disabled*. When enabled, the distance of the bones in stage 3 is constrained using joints in the selected physics engine. This may reduce misrepresentations due to bad connectivity or miscalculations in stage 2.
`maxVelocity`	*3*. Stage 3 physics limit the linear velocity from stage 2 in each direction to the set value.
`scaleVelocities`	*10*. Before limiting to a maxVelocity, the linear velocities from stage 2 are multiplied with the given value.

6.Evaluation

6.1. Considerations

6.1.1. Measurement Techniques

There are two ways of measuring durations, either directly in the code or using the evaluation modules.

Measuring durations in the code does usually not contain the time it takes to send to or receive from Ubi-Interact. It should lead to more uniform and reproducible results. However, the comparability across implementations – not just the two, also potential future implementations – is not necessarily given. Different implementations might fire events and do calculations at different points in time. Comparing measured durations needs to be done very carefully to avoid wrong conclusions.

Measuring durations using the evaluation module includes the time that the communication with the master node takes. Depending on the setup, there may be other traffic that may be prioritized. Thus, results should only be compared with each others when using local servers and measurements should be taken over a longer time frame to avoid artifacts due to network spikes.

One should also keep in mind that all evaluation runs use prerecorded and optimized animation loops. This has the advantage that there is a full pose to compare to – when using controllers, there is no way to programmatically tell whether the calculation was correct. However, using animations might lead to different behavior different than when using real inputs: Noisy data, connectivity issues, a larger range of possible movements and different body types can lead to issues that cannot be caught by this evaluation. Perspectively, it would be insightful to try the software with a tracking suit that tracks all 7 targets.

6.1.2. Sense of Embodiment

High delays lead to a bigger discrepancy between avatar and user pose, which potentially influence the sense of embodiment. However, from low delays, one cannot deduce that a full or high sense of embodiment is given. As described previously, many aspects fall under that term. To make conclusions about the sense of embodiment in a certain configuration, a user study with actual VR hardware would be necessary.

6.2. Desirable Outputs

6.2.1. Delay of Visual Feedback

To create a sense of presence, experiments have shown that a low delay of feedback even more important than pictorial realism [41]. However, there are many other potentially variables that can influence sense of presence and thus sense of embodiment, such as the duration of usage [42] or HMD update rates [43], so literature does usually not give a suggestion of acceptable delays in VR. Welch et al found significant limitation of the sense of presence at an increase of delays from about 200ms to about 1700ms [41]. The delay in a VR application should therefore be significantly lower.

As an observer who looks at somebody else who is embodied, higher delays are justifiable in comparison to the delay of the person who is embodied – the exact value that needs to be achieved differs depending on the exact use case: In a VR sword fight, less delay might be acceptable than in a VR conference. Also one should keep in mind that physics as they are used in this work might dampen fast movements. Smoothened movements could look worse in the data than they would visually appear during actual usage.

There are scenarios in which delayed feedback might even be desirable: Objects may appear heavier when a higher visual delay is applied [44]. In the previous VR sword fight example, this could create a sense of weight of the weapon. However, it could be argued that this kind of desirable delay should be added artificially, so

that it is not bound to specific hardware and network circumstances.

6.2.2. Displacement

This thesis evaluates whether physical embodiment is possible. Because the IK framework has limited features, bigger location displacements are possible. As long as the targets are not displaced by more than a 1-2 centimeters, that does not mean that the output is not suitable. It again depends on the use case whether a pose that looks realistic and points at the right place with his hand, but his knees and elbows are at a wrong location, is correct.

Certainly undesirable are misplaced targets, unrealistic poses and (temporarily) disconnected body parts.

6.3. Evaluation Process

Data of different configurations was collected using the evaluation module. The stage 1 modules were not modified to send a full pose instead of the targets – this evaluation will focus on delays rather than precision of inverse kinematics algorithms.

In the following sections, any combination of web stages (W) and unity stages (U) will be denoted by a three letter combination. For example UWU represents the configuration using the first and third Unity stage, as well as the second web stage. A "*" may denote any of the two implementations.

Evaluation data was recorded using the following process:

1. Modules that were not required were disabled. Additionally, the tag *web* was added to web modules and to the Unity component search requests whenever it helped preventing potential ambiguity.
2. For *WU, a line of unity code was replaced in stage 3, because the lack of angular rotation in the web implementation lead to errors. The necessary change is documented in the web implementation documentation.

3. For *UW, `scaleVelocities` was reduced from 10 to 1, because in the web implementation, smaller values have higher impacts on physics.
4. For U** the animation was started with the space key. W** runs an animation on start.
5. The applications were started, connections were awaited.
6. The evaluation module was started. After about 10 seconds of recording, the animation was stopped using the space key in Unity or the debug option `animation: manual` in the web interface.
7. After a few more seconds, the evaluation module was stopped and the result was copied from the evaluation console.
8. All other modules were stopped completely to ensure that no data from an evaluation run affected the next one.

Unless mentioned differently, the default settings were used in the Unity and Web implementations and Chrome was used to run the main web application and the evaluation module. The evaluation computer specifications are listed in Appendix A. Besides all 8 module combinations, data for WWW was also recorded once using constraints on the physics model and once in Firefox.

6.4. Results and Interpretation

6.4.1. Manual Observations

UUU seemed most realistic and fluent. WWW seemed similarly fluent, but some movements were less realistic due to the missing rotational capabilities of the inverse kinematics library, which was discussed in subsection 5.4.3. It is not surprising that each implementation worked best when used on its own, because that was the main way they were tested during development. Some combinations like UWW worked well.

In general, Unity processing seemed to result in more "overshooting" output from stage 2 – stage 3 avatars would slightly bob up and down.

*WU implementations did not work satisfyingly. The stage 3 avatar would stop moving shortly after the start, in an unrealistic pose. This may be related to the missing rotational data from stage 2 web.

UW* implementations behaved unstable, sometimes they ran well, other times they froze. The reason was not found, but given that the freezing seemed to occasionally stop in Unity while switching windows, it might be related to a a technical detail in the Unity implementation.

In terms of performance, no visible difference was observed with bare eye using Firefox or the additional constraints on the physical model on the evaluation computer. During development on the Laptop, performance in Firefox was observed to be significantly worse than in Chrome.

6.4.2. Observations using the Evaluation Module

All of the following examples refer to the left arm target in comparison to the X-coordinate of the left arm of the physical model. The raw data is available online[1]. In the following graphs, orange represents stage 1 output, while grey is the pose from stage 3. The horizontal axis is time, while the vertical axis represents position in meters.

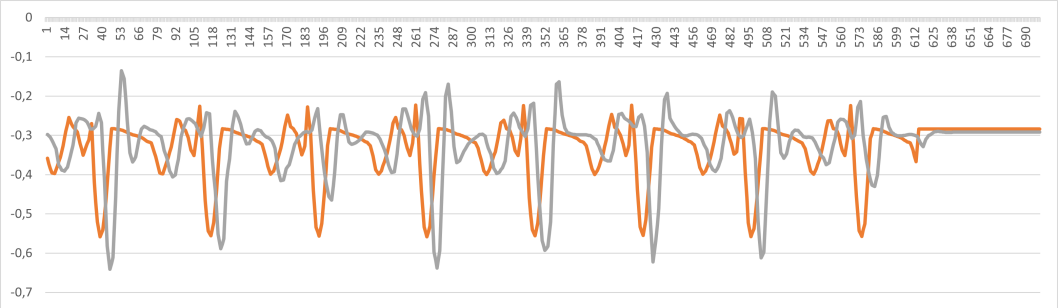


Figure 6.1.: Evaluation of configuration UUU

The presumption that UUU is comparably precise was correct. The previously described effect that the stage 2 module tends to overshoot its results is visible too,

[1]JSON files and conversion scripts at `github.com/goldst/ubii-evaluation-data`

especially after the big gradient, which results from the jump animation.

WWW appears to be similarly performant from what can be read from the data, but a new observation can be made:

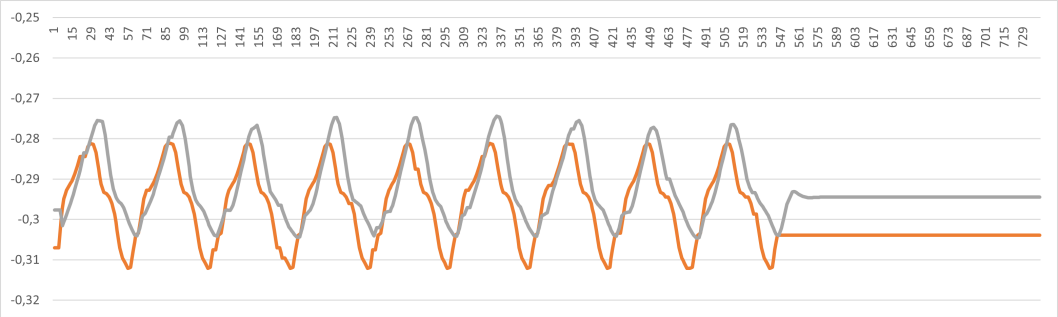


Figure 6.2.: Evaluation of configuration WWW

By default, the web implementation, more specifically the web implementation, displaces the result in the X direction. While the amount of displacement in this direction is not big enough to have a big visual impact in real scenarios, it may contribute to unstable calculations due to incorrect feedback to other modules.

Not in all cases, the data from the evaluation module was a precise representation of the process. In some cases, such as the UWW run, the evaluation module did not receive data during periods where the physical embodiment process seemingly went well. Visually, it might have had low impact, because of the continuous movement while no new velocity data is received. Possible reasons contain the already big amount of data that is being sent over the network, and that the browser could limit tabs in background. Therefore, future evaluation modules should not be created for browsers and should run for a longer time.

7.Outlook

7.1. Potential future work on implementation

The following is a summary of ideas to improve the existing implementation further.

Stage 2 IK needs to be improved. Either additional constraints and rotation data must be added to the existing library, or a different library should be chosen if the performance proves adequate.

Performance does not seem to be the main bottleneck, but if it becomes a problem due to a library switch, porting parts of the application to WebAssembly or increasing the use of multithreading could be considered.

An interesting test would be to use actual inputs instead of recorded ones. This would enable the user to test a more considerable variety of poses but comes at the cost of not being able to compare all bone positions to the actual ones, which are a part of animations.

The evaluation module can save much more data than what was analyzed in the evaluation. This data should be analyzed further to find more weak points, similarities, and differences in the implementations. Additionally, for test purposes, stage 1 could publish not only targets but the whole pose. By doing that, it would be possible to compare the actual pose to the IK pose.

7.2. Potential research beyond the physical embodiment process

The following are some possibilities to continue research in and beyond the topic of web-based physical embodiment using Ubi-Interact.

Even the Unity implementation could be considered a high level implementation - for applications in which performance is crucial, an implementation that compiles

to machine code could be beneficial. However, it is questionable to what extent the performance could be improved in that way because the performance bottleneck likely stems from local network communication.

The bone rig used in this thesis is equivalent to the one in the Unity implementation, a human rig. Often, physical embodiment requires other avatars. For example, some applications might only want to embody a more exact model of the user's hand. Furthermore, applications in which the user is a different animal, for example, a dog, are thinkable. This would require using a different avatar too.

There are many users and avatars in many scenarios requiring physical embodiment. While performance seems not to be a big issue on a simple implementation with one avatar, this might be different in such applications. It should be evaluated separately whether browsers using Ubi-Interact are a suitable solution to such problems.

The avatar being physics-based opens many possibilities for interaction with virtual objects. For example, an avatar might trip over a stone that only exists virtually. In Human-Computer Interaction, such situations could be explored further. How does a human react if the avatar falls due to a virtual stone? How significant can the discrepancy between the avatar and his pose be without the user feeling disconnected? This also plays a role due to possible delays because of low network speed.

Finally, avatars do not have to be virtual. In the field of robotics, there are several examples of humanoid robots. On the one hand, replacing the virtual representation with a robotic one could result in a different set of problems because the forces that must be applied might differ significantly. On the other hand, this would potentially open many possibilities for using Ubi-Interact across other fields of research.

8.Conclusion

Physical embodiment with Ubi-Interact is viable in Browsers. This thesis showed that all parts of the process could run in browsers. Reusable modules that serve as a base for physical embodiment on the web were developed and are available as open-source software for usage and modification.

Furthermore, it was shown that communication between the web and Unity implementations is possible. Thus, any combination of modules across implementations can be used to realize the physical embodiment process with some adjustments. However, the implicit nature of parts of the projects often made compatibility hard. Many details that indirectly come from IK and physics implementations might be taken for granted in one implementation but work differently in the other. More documentation and a precise protocol that all implementations shall follow may simplify the process in the future.

During development and evaluation, some caveats for physical embodiment in browsers were found that should be considered when planning the development of applications that depend on this process:

Technical limitations exist. One example is the typically ungraceful application exits in browsers. In cases such as closing or reloading the tab, it is impossible to properly disconnect Ubi-Interact, which can lead to incorrect component data at the next load.

However, technical limitations are less severe than libraries' unavailability, which may be related to the fact that browser environments are still uncommon as platforms for 3D software, such as games. The libraries created during this thesis are a first step towards overcoming these limitations. More work will be necessary to create stable and feature-complete libraries, especially in Inverse Kinematics.

This thesis also illustrated how versatile Ubi-Interact could be used. Devices can not only be used for simulating real devices or representing an entity in a visual process; in this project, a device also served the purpose of data analysis. Many more use cases are conceivable.

It was illustrated that the projects developed in this thesis are prototypes and bases for potential future work. The topics of this thesis cover a wide range of technologies and ideas that could be used to improve the physical embodiment process further.

A.General Addenda

A.1. List of bone names used across implementations

Extracted from ubii-vr-physics-embodiment-unity [4].

`Hips`, `LeftUpperLeg`, `RightUpperLeg`, `LeftLowerLeg`, `RightLowerLeg`,
`LeftFoot`, `RightFoot`, `Spine`, `Chest`, `Neck`, `Head`,
`LeftShoulder`, `RightShoulder`, `LeftUpperArm`, `RightUpperArm`,
`LeftLowerArm`, `RightLowerArm`, `LeftHand`, `RightHand`, `LeftToes`,
`RightToes`, `LeftThumbProximal`, `LeftThumbIntermediate`, `LeftThumbDistal`,
`LeftIndexProximal`, `LeftIndexIntermediate`, `LeftIndexDistal`,
`LeftMiddleProximal`, `LeftMiddleIntermediate`, `LeftMiddleDistal`,
`LeftRingProximal`, `LeftRingIntermediate`, `LeftRingDistal`, `LeftLittleProximal`,
`LeftLittleIntermediate`, `LeftLittleDistal`, `RightThumbProximal`,
`RightThumbIntermediate`, `RightThumbDistal`, `RightIndexProximal`,
`RightIndexIntermediate`, `RightIndexDistal`, `RightMiddleProximal`,
`RightMiddleIntermediate`, `RightMiddleDistal`, `RightRingProximal`,
`RightRingIntermediate`, `RightRingDistal`, `RightLittleProximal`,
`RightLittleIntermediate`, `RightLittleDistal`, `UpperChest`

A.2. Computer specifications

A.2.1. Desktop computer, used for evaluations

Windows 10 Enterprise LTSC, 64bit
Intel(R) Core(TM) i7-8086K CPU @ 4.00GHz
GeForce GTX 1080
16.0 GB RAM

A.2.2. Laptop, for further observations

Arch Linux 5.18.3-arch1-1, 64bit
Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz,
builtin GPU
8 GB RAM

B.Figures

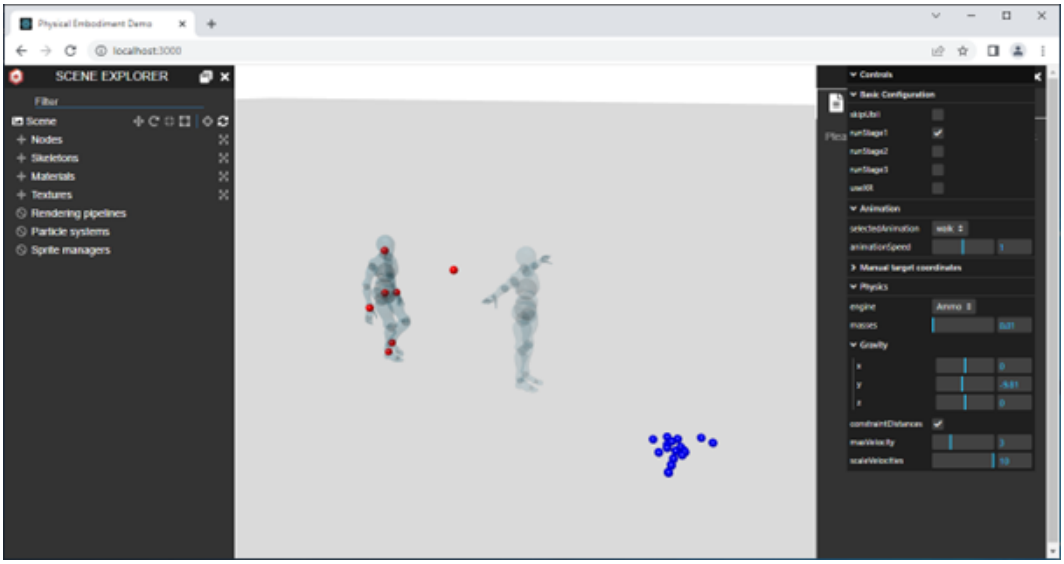


Figure B.1.: Screenshot of the web implementation. Only stage 1 is enabled, the other two stage avatars are in a unused state. The debugging UI on the right was used to make this configuration, along with enabling `constraintDistances`, which has no effect due to the disabled stage 3. The Scene Explorer on the left was kept in the application to allow examining the skeleton's structure.

List of Figures

List of Tables

Bibliography

[1] S. Weber, M. Ludwig, and G. Klinker. "Ubi-Interact: A modular approach to connecting systems". In: *EAI Endorsed Transactions on Mobile Communications and Applications* 6.19 (2021), e5.

[2] S. Weber and G. Klinker. "Vr re-embodiment in the neurorobotics platform". In: *Mensch und Computer 2019-Workshopband* (2019).

[3] B. Maclntyre and T. F. Smith. "Thoughts on the Future of WebXR and the Immersive Web". In: *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 2018, pp. 338–342. DOI: 10.1109/ISMAR-Adjunct.2018.00099.

[4] S. Weber. *ubii-vr-physics-embodiment-unity. Github repository*. 2022. URL: https://github.com/SandroWeber/ubii-vr-physics-embodiment-unity (visited on 02/24/2022).

[5] *embody*. In: *Cambridge Academic Content Dictionary*. Cambridge University Press. URL: https://dictionary.cambridge.org/dictionary/english/embody (visited on 05/09/2022).

[6] B. Spanlang, J.-M. Normand, D. Borland, K. Kilteni, E. Giannopoulos, A. Pomés, M. González-Franco, D. Perez-Marcos, J. Arroyo-Palacios, X. N. Muncunill, and M. Slater. "How to Build an Embodiment Lab: Achieving Body Representation Illusions in Virtual Reality". In: *Frontiers in Robotics and AI* 1 (2014). ISSN: 2296-9144. DOI: 10.3389/frobt.2014.00009. URL: https://www.frontiersin.org/article/10.3389/frobt.2014.00009.

[7] E. Negri, L. Fumagalli, and M. Macchi. "A review of the roles of digital twin in CPS-based production systems". In: *Procedia Manufacturing* 11 (2017), pp. 939–948.

[8] K. Kilteni, R. Groten, and M. Slater. "The sense of embodiment in virtual reality". In: *Presence: Teleoperators and Virtual Environments* 21.4 (2012), pp. 373–387.

[9] D. Kasprowicz and S. Rieger. *Handbuch Virtualität*. Springer, 2020.

[10] K. M. Besmer. "What robotic re-embodiment reveals about virtual re-embodiment". In: *Postphenomenological Investigations: Essays on Human-Technology Relations* (2015), pp. 55–72.

[11] M. Luria, S. Reig, X. Z. Tan, A. Steinfeld, J. Forlizzi, and J. Zimmerman. "Re-Embodiment and Co-Embodiment: Exploration of social presence for robots and conversational agents". In: *Proceedings of the 2019 on Designing Interactive Systems Conference*. 2019, pp. 633–644.

[12] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. "Inverse kinematics techniques in computer graphics: A survey". In: *Computer graphics forum*. Vol. 37. 6. Wiley Online Library. 2018, pp. 35–58.

[13] J. Collins, S. Chand, A. Vanderkop, and D. Howard. "A review of physics simulators for robotic applications". In: *IEEE Access* (2021).

[14] T. Geijtenbeek, N. Pronost, A. Egges, and M. H. Overmars. "Interactive Character Animation using Simulated Physics." In: *Eurographics (State of the Art Reports)* (2011), pp. 127–149.

[15] *Inverse Kinematics*. In: *Unity User Manual 2021.3 (LTS)*. URL: `https://docs.unity3d.com/Manual/InverseKinematics.html` (visited on 05/09/2022).

[16] *Built-in 3D Physics*. In: *Unity User Manual 2021.3 (LTS)*. URL: `https://docs.unity3d.com/Manual/PhysicsOverview.html` (visited on 05/09/2022).

[17] S. Weber, D. Dyrda, ate362, M. Lohr, baumlos, and L. Goldstein. *ubii-msg-formats. Github repository*. 2022. URL: `https://github.com/SandroWeber/ubii-msg-formats` (visited on 02/24/2022).

[18] F. Brizzi, L. Peppoloni, A. Graziano, E. Di Stefano, C. A. Avizzano, and E. Ruffaldi. "Effects of augmented reality on the performance of teleoperated industrial assembly tasks in a robotic embodiment". In: *IEEE Transactions on Human-Machine Systems* 48.2 (2017), pp. 197–206.

[19] H. Laaki, Y. Miche, and K. Tammi. "Prototyping a digital twin for real time remote control over mobile networks: Application of remote surgery". In: *Ieee Access* 7 (2019), pp. 20325–20336.

[20] C. Greenhalgh and S. Benford. "Virtual reality tele-conferencing: Implementation and experience". In: *Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work ECSCW'95*. Springer. 1995, pp. 165–180.

[21] P. D. Pazour, A. Janecek, and H. Hlavacs. "Virtual reality conferencing". In: *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*. IEEE. 2018, pp. 84–91.

[22] VRChat Inc. *VRChat*. URL: `https://store.steampowered.com/app/438100/VRChat/` (visited on 05/09/2022).

[23] M. Rzeszewski and L. Evans. "Virtual place during quarantine–a curious case of VRChat". In: *Rozwój Regionalny i Polityka Regionalna* 51 (2020), pp. 57–75.

[24] C. Montemorano. "Body Language: Avatars, Identity Formation, and Communicative Interaction in VRChat". In: (2020).

[25] A. S. Kim. "Virtual Worldmaking: A Phantasmal Media Approach to VRChat". PhD thesis. Massachusetts Institute of Technology, 2021.

[26] *Working with JavaScript*. 2022. URL: https://code.visualstudio.com/docs/nodejs/working-with-javascript (visited on 02/24/2022).

[27] Stoplion, J. de Jong, and abhishekmatta999. *josdejong/workerpool. Possible to import libraries into Worker?* URL: https://github.com/josdejong/workerpool/issues/47 (visited on 05/09/2022).

[28] M. Kruisselbrink. *File System Access. Draft Community Group Report, 11 April 2022*. Web Platform Incubator Community Group, 2022. URL: https://wicg.github.io/file-system-access/ (visited on 05/09/2022).

[29] A. Deveria. *File System Access API. Can I use... Support tables for HTML5, CSS3, etc*. 2022. URL: https://caniuse.com/native-filesystem-api (visited on 05/09/2022).

[30] Mozilla. *Mozilla specification propositions. File System Access*. 2022. URL: https://mozilla.github.io/standards-positions/#native-file-system (visited on 05/09/2022).

[31] StatCounter. *Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2021 [Graph]*. 2022. URL: https://www-statista-com.eaccess.ub.tum.de/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ (visited on 02/24/2022).

[32] R. Weber, PirateJC, idutta2007, and Popov72. *Babylon.js Documentation. Bones and Skeletons*. URL: https://doc.babylonjs.com/divingDeeper/mesh/bonesSkeletons#boneikcontroller (visited on 05/09/2022).

[33] J. Friend. *BussIK-js. GitHub repository*. https://github.com/jsdf/BussIK-js. 2022. (Visited on 05/09/2022).

[34] S. R. Buss and J.-S. Kim. "Selectively damped least squares for inverse kinematics". In: *Journal of Graphics tools* 10.3 (2005), pp. 37–49.

[35] G. Johnson. *closed-chain-ik-js. GitHub repository*. https://github.com/gkjohnson/closed-chain-ik-js. 2022. (Visited on 05/09/2022).

[36] lo-th. *Fullik. GitHub repository*. `https://github.com/lo-th/fullik`. 2022. (Visited on 05/09/2022).

[37] A. Lansley, P. Vamplew, P. Smith, and C. Foale. "Caliko: An inverse kinematics software library implementation of the FABRIK algorithm". In: *Journal of Open Research Software* 4.1 (2016).

[38] A. Aristidou and J. Lasenby. "FABRIK: A fast, iterative solver for the Inverse Kinematics problem". In: *Graphical Models* 73.5 (2011), pp. 243–260.

[39] R. Eisele. *Proof: Quaternion from two vectors*. URL: `https://www.xarg.org/proof/quaternion-from-two-vectors/` (visited on 05/09/2022).

[40] Mozilla Mixed Reality. *WebXR API Emulator*. URL: `https://addons.mozilla.org/de/firefox/addon/webxr-api-emulator/` (visited on 05/09/2022).

[41] R. B. Welch, T. T. Blackmon, A. Liu, B. A. Mellers, and L. W. Stark. "The effects of pictorial realism, delay of visual feedback, and observer interactivity on the subjective sense of presence". In: *Presence: Teleoperators & Virtual Environments* 5.3 (1996), pp. 263–273.

[42] L. C. Van Dam and J. R. Stephens. "Effects of prolonged exposure to feedback delay on the qualitative subjective experience of virtual reality". In: *PloS one* 13.10 (2018), e0205145.

[43] M. P. Snow. "Charting presence in virtual environments and its effects on performance". PhD thesis. Virginia Polytechnic Institute and State University, 1996.

[44] V. van Polanen, R. Tibold, A. Nuruki, and M. Davare. "Visual delay affects force scaling and weight perception during object lifting in virtual reality". In: *Journal of neurophysiology* 121.4 (2019), pp. 1398–1409.